**2024年秋季学习课后作业**

第1次作业

1. P42-P43 1 5 6
2. 计算点（1，2，3）和点（3，4，8）的L1、L2、L∞距离。

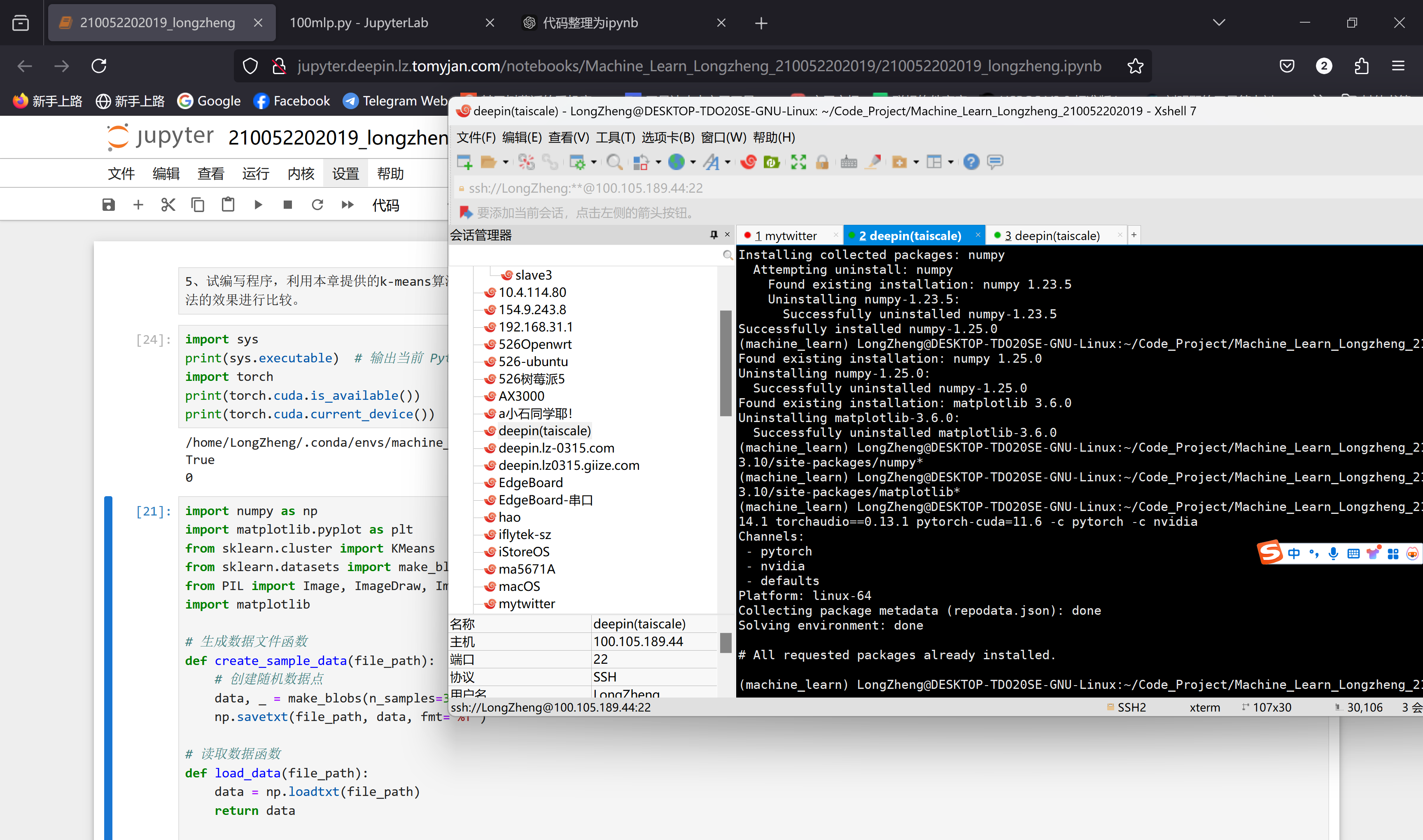
L1 距离（曼哈顿距离）：

L2 距离（欧几里得距离）：

L∞ 距离（切比雪夫距离）：

1. 试编写程序，利用本章提供的k-means算法代码或者sklearn.cluster.KMeans算法函数实现二分k-means算法，对随书资源中的kmeansSamples.txt文件中的点进行分簇，并于k-means算法的效果进行比较。

首先进行环境的搭建：



import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# 生成数据文件函数

def create\_sample\_data(file\_path):

    # 创建随机数据点

    data, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)

    np.savetxt(file\_path, data, fmt='%f')

# 读取数据函数

def load\_data(file\_path):

    data = np.loadtxt(file\_path)

    return data

# 二分 K-means 算法实现

def bi\_kmeans(data, k):

    m = data.shape[0]

    cluster\_assessment = np.zeros((m, 2))  # 每个样本的簇分配及误差

    # 初始簇为所有数据

    centroid = np.mean(data, axis=0).tolist()

    centroids = [centroid]

    # 计算所有数据点到质心的平方误差

    for j in range(m):

        cluster\_assessment[j, 1] = np.sum((data[j] - centroid) \*\* 2)

    while len(centroids) < k:

        lowest\_sse = float('inf')

        # 尝试分裂每一个簇

        for i in range(len(centroids)):

            points\_in\_cluster = data[np.nonzero(cluster\_assessment[:, 0] == i)[0], :]

            if len(points\_in\_cluster) <= 1:  # 如果簇中只有一个点，跳过分裂

                continue

            kmeans = KMeans(n\_clusters=2, random\_state=42).fit(points\_in\_cluster)

            centroids\_split = kmeans.cluster\_centers\_

            cluster\_ass\_split = kmeans.labels\_

            sse\_split = np.sum((points\_in\_cluster - centroids\_split[cluster\_ass\_split]) \*\* 2)

            sse\_not\_split = np.sum(cluster\_assessment[np.nonzero(cluster\_assessment[:, 0] != i)[0], 1])

            # 计算新的总误差

            if (sse\_split + sse\_not\_split) < lowest\_sse:

                best\_centroid\_to\_split = i

                best\_new\_centroids = centroids\_split

                best\_cluster\_ass = cluster\_ass\_split.copy()

                lowest\_sse = sse\_split + sse\_not\_split

        # 更新簇质心

        best\_cluster\_ass = best\_cluster\_ass.astype(int)

        centroids[best\_centroid\_to\_split] = best\_new\_centroids[0, :].tolist()

        centroids.append(best\_new\_centroids[1, :].tolist())

        # 更新簇分配结果

        indices = np.nonzero(cluster\_assessment[:, 0] == best\_centroid\_to\_split)[0]

        for idx, label in zip(indices, best\_cluster\_ass):

            cluster\_assessment[idx, 0] = label + len(centroids) - 2

            cluster\_assessment[idx, 1] = np.sum((data[idx] - best\_new\_centroids[label]) \*\* 2)

    return np.array(centroids), cluster\_assessment

# 主程序

def main():

    # 创建数据文件

    file\_path = "kmeansSamples.txt"

    create\_sample\_data(file\_path)

    # 加载数据

    data = load\_data(file\_path)

    # 运行标准 K-means 算法

    k = 4  # 假设我们要分为 4 个簇

    kmeans = KMeans(n\_clusters=k, random\_state=42).fit(data)

    kmeans\_labels = kmeans.labels\_

    # 运行二分 K-means 算法

    centroids, bi\_kmeans\_assessment = bi\_kmeans(data, k)

    bi\_kmeans\_labels = bi\_kmeans\_assessment[:, 0]

    # 画出 K-means 结果

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)

    plt.scatter(data[:, 0], data[:, 1], c=kmeans\_labels, cmap='viridis')

    plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', marker='x')

    plt.title("K-means Clustering")

    # 画出二分 K-means 结果

    plt.subplot(1, 2, 2)

    plt.scatter(data[:, 0], data[:, 1], c=bi\_kmeans\_labels, cmap='viridis')

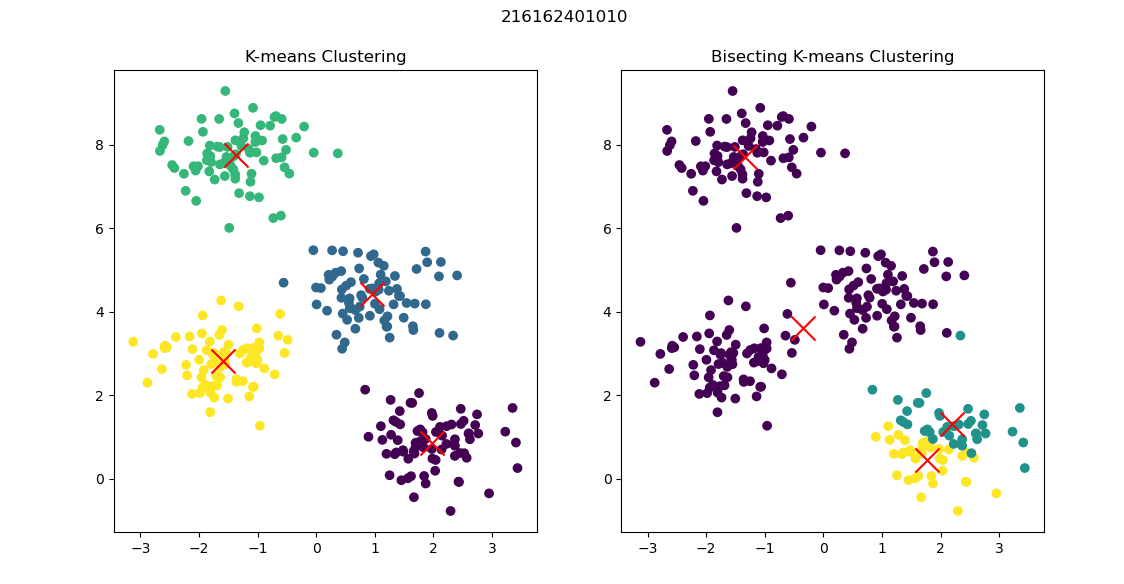
    plt.scatter(np.array(centroids)[:, 0], np.array(centroids)[:, 1], s=300, c='red', marker='x')

    plt.title("Bisecting K-means Clustering")

    plt.show()

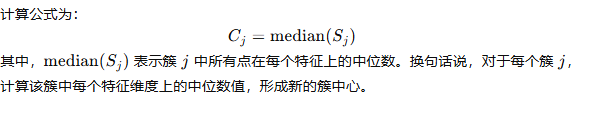
if \_\_name\_\_ == "\_\_main\_\_":

    main()



210052202019 LongZheng

1. 在k-means算法中，当采用曼哈顿距离作为距离度量方法、以距离和作为损失函数时，称为k-median算法。
2. 簇中心如何计算？



1. 修改本章提供的k-means算法代码实现该算法。

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# 生成数据文件函数

def create\_sample\_data(file\_path):

    # 创建随机数据点

    data, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)

    np.savetxt(file\_path, data, fmt='%f')

# 读取数据函数

def load\_data(file\_path):

    data = np.loadtxt(file\_path)

    return data

# K-median 算法实现

def k\_median(data, k, max\_iter=300):

    m, n = data.shape

    # 随机选择初始簇中心

    centroids = data[np.random.choice(m, k, replace=False)]

    labels = np.zeros(m)

    for \_ in range(max\_iter):

        # 计算每个点到所有质心的曼哈顿距离，并选择最近的质心

        distances = np.sum(np.abs(data[:, np.newaxis] - centroids), axis=2)

        new\_labels = np.argmin(distances, axis=1)

        # 如果簇分配没有变化，则退出

        if np.array\_equal(labels, new\_labels):

            break

        labels = new\_labels

        # 更新簇中心为当前簇中所有点的中位数

        for i in range(k):

            points\_in\_cluster = data[labels == i]

            if len(points\_in\_cluster) > 0:

                centroids[i] = np.median(points\_in\_cluster, axis=0)

    return centroids, labels

# 主程序

def main():

    # 创建数据文件

    file\_path = "kmeansSamples.txt"

    create\_sample\_data(file\_path)

    # 加载数据

    data = load\_data(file\_path)

    # 运行标准 K-means 算法

    k = 4  # 假设我们要分为 4 个簇

    kmeans = KMeans(n\_clusters=k, random\_state=42).fit(data)

    kmeans\_labels = kmeans.labels\_

    # 运行 K-median 算法

    k\_median\_centroids, k\_median\_labels = k\_median(data, k)

    # 画出 K-means 结果

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)

    plt.scatter(data[:, 0], data[:, 1], c=kmeans\_labels, cmap='viridis')

    plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', marker='x')

    plt.title("K-means Clustering")

    # 画出 K-median 结果

    plt.subplot(1, 2, 2)

    plt.scatter(data[:, 0], data[:, 1], c=k\_median\_labels, cmap='viridis')

    plt.scatter(k\_median\_centroids[:, 0], k\_median\_centroids[:, 1], s=300, c='red', marker='x')

    plt.title("K-median Clustering")

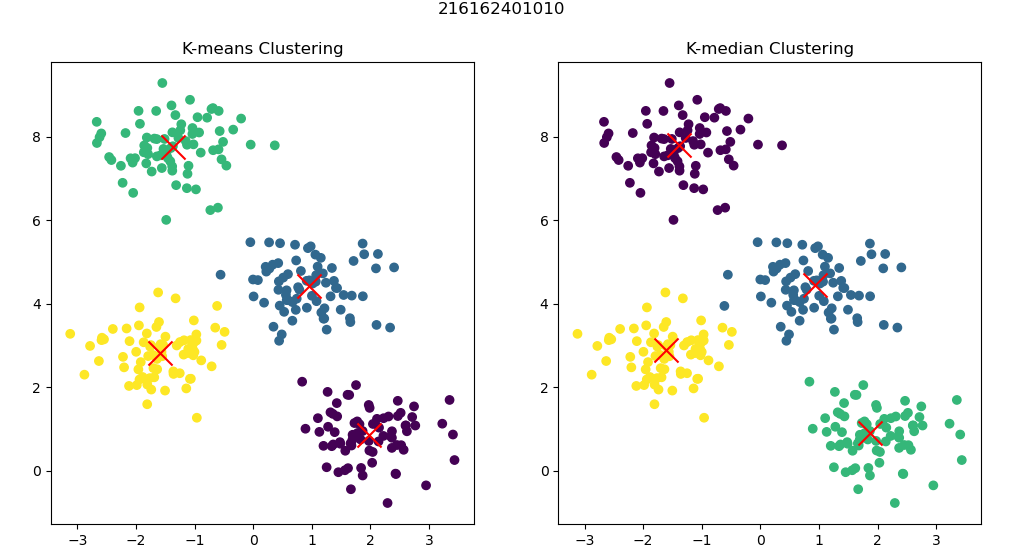
    plt.suptitle("210052202019 LongZheng")

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

210052202019 LongZheng



1. P78-P79 1 3 4
2. 用sklearn.linear\_model包中的LinearRegression对表3-1所示的示例进行线性回归实验，比较结果。

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# 表 3-1 数据

X = np.array([1.0, 2.0, 3.0, 4.0, 5.0]).reshape(-1, 1)

y = np.array([1.5, 2.8, 3.9, 5.2, 7.4])

# 创建线性回归模型

model = LinearRegression()

model.fit(X, y)

# 预测结果

y\_pred = model.predict(X)

# 模型评估

mse = mean\_squared\_error(y, y\_pred)

r2 = r2\_score(y, y\_pred)

# 打印结果

print("模型系数 (Slope):", model.coef\_[0])

print("截距 (Intercept):", model.intercept\_)

print("均方误差 (MSE):", mse)

print("R^2 值:", r2)

# 创建结果比较表格

data = {

    'X': X.flatten(),

    'y\_实际值': y,

    'y\_预测值': y\_pred

}

result\_df = pd.DataFrame(data)

print("\n结果比较表格:\n", result\_df)

# 可视化结果

plt.scatter(X, y, color='blue', label='实际值')

plt.plot(X, y\_pred, color='red', linewidth=2, label='预测值')

plt.xlabel('X')

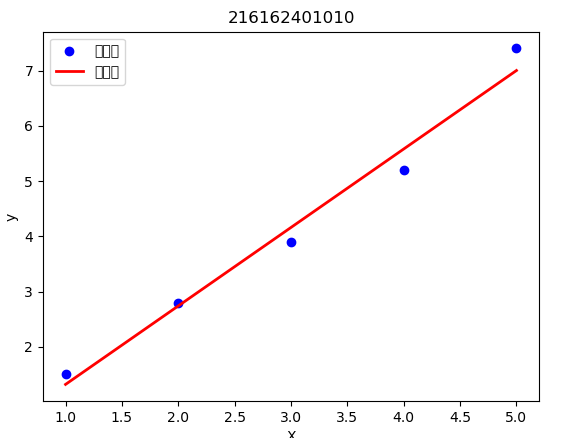
plt.ylabel('y')

plt.legend()

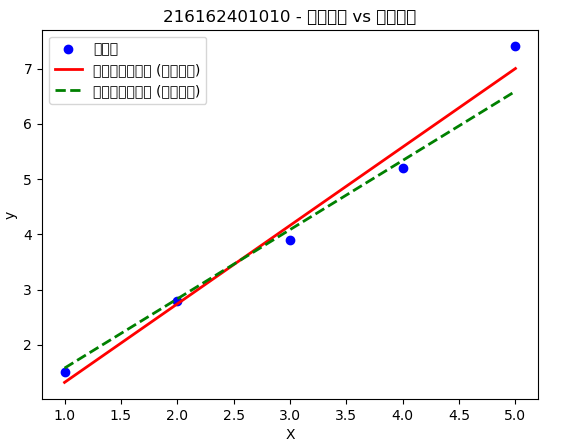
plt.title('210052202019 LongZheng')

plt.show()

210052202019 LongZheng



1. 查阅资料，研究梯度下降法中步长的动态调整方法，试将代码3-6中固定步长改为动态步长，并对比两者运行结果。



210052202019 LongZheng

1. 试修改代码3-6实现批梯度下降和随机梯度下降算法，并从时间和结果两方面与原算法进行比较。

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import time

# Table 3-1 Data

X = np.array([1.0, 2.0, 3.0, 4.0, 5.0]).reshape(-1, 1)

y = np.array([1.5, 2.8, 3.9, 5.2, 7.4])

# Create Linear Regression Model

model = LinearRegression()

start\_time = time.time()

model.fit(X, y)

end\_time = time.time()

# Prediction Results

y\_pred = model.predict(X)

# Model Evaluation

mse = mean\_squared\_error(y, y\_pred)

r2 = r2\_score(y, y\_pred)

# Print Results

print("模型系数 (Slope):", model.coef\_[0])

print("截距 (Intercept):", model.intercept\_)

print("均方误差 (MSE):", mse)

print("R^2 值:", r2)

print("线性回归耗时 (s):", end\_time - start\_time)

# Create Result Comparison Table

data = {

    'X': X.flatten(),

    'y\_Actual Values': y,

    'y\_预测值': y\_pred

}

result\_df = pd.DataFrame(data)

print("\n结果比较表格:\n", result\_df)

# Visualize Results

plt.scatter(X, y, color='blue', label='实际值')

plt.plot(X, y\_pred, color='red', linewidth=2, label='预测值')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.title('210052202019 LongZheng')

plt.show()

# Dynamic Step Size Gradient Descent Implementation

def gradient\_descent(X, y, lr=0.01, epochs=1000, tolerance=1e-6):

    m = 0  # 初始斜率

    b = 0  # 初始截距

    n = len(y)

    prev\_cost = float('inf')

    for epoch in range(epochs):

        y\_pred = m \* X + b

        residuals = y - y\_pred.flatten()

        dm = -2/n \* sum(X.flatten() \* residuals)

        db = -2/n \* sum(residuals)

        # 动态调整步长

        lr = lr / (1 + 0.01 \* epoch)

        # 更新参数

        m -= lr \* dm

        b -= lr \* db

        # 计算当前损失

        cost = mean\_squared\_error(y, y\_pred.flatten())

        if abs(prev\_cost - cost) < tolerance:

            break

        prev\_cost = cost

    return m, b

# Using Gradient Descent

start\_time = time.time()

m, b = gradient\_descent(X, y)

end\_time = time.time()

# 使用梯度下降法预测结果

y\_pred\_gd = m \* X + b

# 模型评估

mse\_gd = mean\_squared\_error(y, y\_pred\_gd)

r2\_gd = r2\_score(y, y\_pred\_gd)

# 打印梯度下降法结果

print("Gradient Descent Model Coefficient (Slope):", m)

print("Gradient Descent Intercept:", b)

print("Gradient Descent Mean Squared Error (MSE):", mse\_gd)

print("Gradient Descent R^2 Value:", r2\_gd)

print("Gradient Descent Time (s):", end\_time - start\_time)

# 创建结果比较表格

data\_gd = {

    'X': X.flatten(),

    'y\_实际值': y,

    'y\_预测值\_梯度下降': y\_pred\_gd.flatten()

}

result\_df\_gd = pd.DataFrame(data\_gd)

print("\nGradient Descent Result Comparison Table:\n", result\_df\_gd)

# 可视化比较结果

plt.scatter(X, y, color='blue', label='实际值')

plt.plot(X, y\_pred, color='red', linewidth=2, label='Linear Regression Prediction (Fixed Step Size)')

plt.plot(X, y\_pred\_gd, color='green', linestyle='--', linewidth=2, label='Gradient Descent Prediction (Dynamic Step Size)')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.title('210052202019 LongZheng - 固定步长 vs 动态步长')

plt.show()

# Batch Gradient Descent Implementation

def batch\_gradient\_descent(X, y, lr=0.01, epochs=1000, tolerance=1e-6):

    m = 0  # 初始斜率

    b = 0  # 初始截距

    n = len(y)

    prev\_cost = float('inf')

    for epoch in range(epochs):

        y\_pred = m \* X + b

        residuals = y - y\_pred.flatten()

        dm = -2/n \* sum(X.flatten() \* residuals)

        db = -2/n \* sum(residuals)

        # 更新参数

        m -= lr \* dm

        b -= lr \* db

        # 计算当前损失

        cost = mean\_squared\_error(y, y\_pred.flatten())

        if abs(prev\_cost - cost) < tolerance:

            break

        prev\_cost = cost

    return m, b

# Using Batch Gradient Descent

start\_time = time.time()

m\_batch, b\_batch = batch\_gradient\_descent(X, y)

end\_time = time.time()

# 使用批梯度下降预测结果

y\_pred\_batch = m\_batch \* X + b\_batch

# 模型评估

mse\_batch = mean\_squared\_error(y, y\_pred\_batch)

r2\_batch = r2\_score(y, y\_pred\_batch)

# 打印批梯度下降法结果

print("Batch Gradient Descent Model Coefficient (Slope):", m\_batch)

print("Batch Gradient Descent Intercept:", b\_batch)

print("Batch Gradient Descent Mean Squared Error (MSE):", mse\_batch)

print("Batch Gradient Descent R^2 Value:", r2\_batch)

print("Batch Gradient Descent Time (s):", end\_time - start\_time)

# Stochastic Gradient Descent Implementation

def stochastic\_gradient\_descent(X, y, lr=0.01, epochs=1000, tolerance=1e-6):

    m = 0  # 初始斜率

    b = 0  # 初始截距

    n = len(y)

    prev\_cost = float('inf')

    for epoch in range(epochs):

        for i in range(n):

            xi = X[i]

            yi = y[i]

            y\_pred = m \* xi + b

            residual = yi - y\_pred

            dm = -2 \* xi \* residual

            db = -2 \* residual

            # 更新参数

            m -= lr \* dm

            b -= lr \* db

        # 计算当前损失

        y\_pred\_all = m \* X + b

        cost = mean\_squared\_error(y, y\_pred\_all.flatten())

        if abs(prev\_cost - cost) < tolerance:

            break

        prev\_cost = cost

    return m, b

# Using Stochastic Gradient Descent

start\_time = time.time()

m\_sgd, b\_sgd = stochastic\_gradient\_descent(X, y)

end\_time = time.time()

# 使用随机梯度下降预测结果

y\_pred\_sgd = m\_sgd \* X + b\_sgd

# 模型评估

mse\_sgd = mean\_squared\_error(y, y\_pred\_sgd)

r2\_sgd = r2\_score(y, y\_pred\_sgd)

# 打印随机梯度下降法结果

print("Stochastic Gradient Descent Model Coefficient (Slope):", m\_sgd)

print("Stochastic Gradient Descent Intercept:", b\_sgd)

print("Stochastic Gradient Descent Mean Squared Error (MSE):", mse\_sgd)

print("Stochastic Gradient Descent R^2 Value:", r2\_sgd)

print("Stochastic Gradient Descent Time (s):", end\_time - start\_time)

# 可视化批梯度下降和随机梯度下降结果

plt.scatter(X, y, color='blue', label='实际值')

plt.plot(X, y\_pred, color='red', linewidth=2, label='线性回归预测值 (固定步长)')

plt.plot(X, y\_pred\_gd, color='green', linestyle='--', linewidth=2, label='梯度下降预测值 (动态步长)')

plt.plot(X, y\_pred\_batch, color='orange', linestyle='-.', linewidth=2, label='Batch Gradient Descent Prediction')

plt.plot(X, y\_pred\_sgd, color='purple', linestyle=':', linewidth=2, label='Stochastic Gradient Descent Prediction')

plt.xlabel('X')

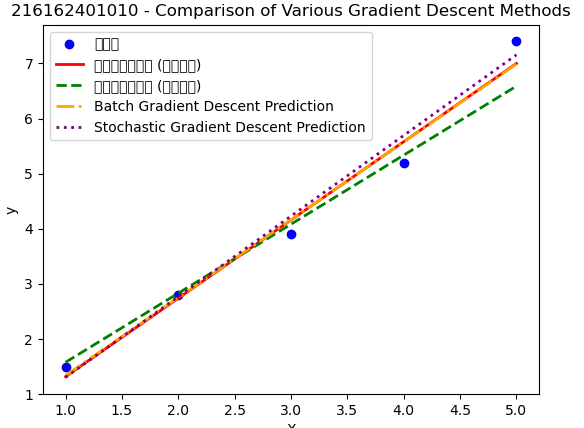
plt.ylabel('y')

plt.legend()

plt.title('210052202019 - Comparison of Various Gradient Descent Methods')

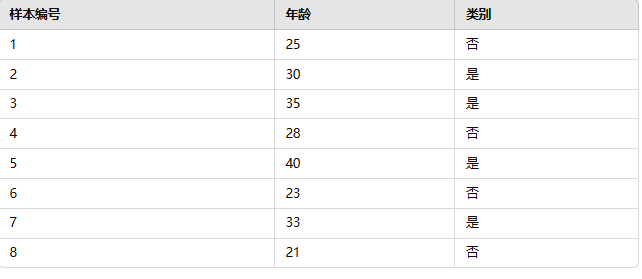
plt.show()

210052202019 - Comparison of Various Gradient Descent Methods



第2次作业

1. P131 1 5 6 7
2. 将表4-1所示的样本集合按年龄（大于等于29）进行切分，试计算切分后的信息增益和基尼指数。



· **信息增益**：0.954

· **切分后的基尼指数**：0

1. 表4-6为某二分类器预测结果的混淆矩阵，试计算准确率、平均准确率、精确率、召回率和F1—score。

表 4-6 某二分类器预测结果

|  |  |  |
| --- | --- | --- |
|  | 预测为“0”的样本数 | 预测为“1”的样本数 |
| 标签为“0”的样本数 | 1026 | 1101 |
| 标签为“1”的样本数 | 1007 | 911026 |

Daan:准确率 (Accuracy)：0.9977

平均准确率 (Balanced Accuracy)：0.7409

精确率 (Precision)：0.9988

召回率 (Recall)：0.9989

F1-score：0.9988

1. 针对随书资源ellipseSamples.txt文件中的二分类样本点，设计逻辑回归模型，进行训练，画出决策边界。

# 210052202019 LongZheng

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import make\_classification

# 1. 生成二分类数据

X, y = make\_classification(n\_samples=100, n\_features=2, n\_informative=2, n\_redundant=0, random\_state=42)

# 将数据保存为txt文件

np.savetxt('classification\_data.txt', np.column\_stack((X, y)), fmt='%f', header='Feature1 Feature2 Label')

# 2. 训练逻辑回归模型

model = LogisticRegression()

model.fit(X, y)

# 3. 可视化数据和决策边界

plt.figure(figsize=(10, 6))

# 绘制样本点

plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='Class 0')

plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='Class 1')

# 绘制决策边界

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 100), np.linspace(y\_min, y\_max, 100))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.5, cmap=plt.cm.Paired)

plt.xlabel('Feature 1')

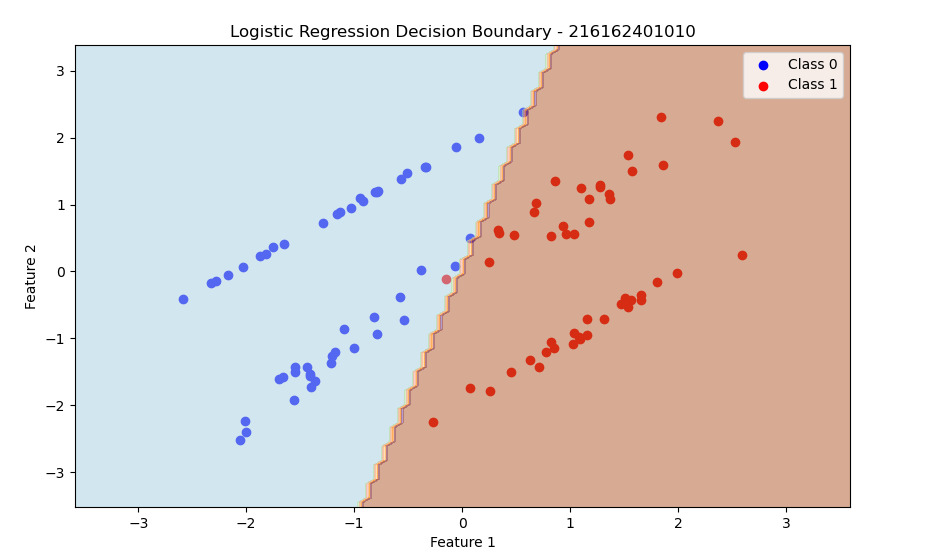
plt.ylabel('Feature 2')

plt.legend()

plt.title('Logistic Regression Decision Boundary - 210052202019 LongZheng')

plt.show()

Logistic Regression Decision Boundary - 210052202019 LongZheng



1. 查阅资料学习sklearn.ensemble.ExtraTreesClassifier的应用方法，并用它来完成优惠券使用示例实验。

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.metrics import classification\_report, accuracy\_score

import matplotlib.pyplot as plt

# 第一步：准备实验环境和数据

# 生成模拟数据：假设数据集有以下特征：

# 用户年龄、是否为新用户、优惠券金额、优惠券有效期、购买习惯评分

np.random.seed(42)

data\_size = 1000

data = {

    'age': np.random.randint(18, 60, data\_size),

    'new\_user': np.random.randint(0, 2, data\_size),

    'coupon\_amount': np.random.randint(5, 100, data\_size),

    'validity\_days': np.random.randint(1, 30, data\_size),

    'purchase\_score': np.random.uniform(0, 1, data\_size),

    'coupon\_used': np.random.randint(0, 2, data\_size)  # 标签：是否使用优惠券

}

df = pd.DataFrame(data)

# 特征和标签

X = df.drop(columns=['coupon\_used'])

y = df['coupon\_used']

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 第二步：使用ExtraTreesClassifier进行训练

# 创建ExtraTreesClassifier实例

et\_classifier = ExtraTreesClassifier(n\_estimators=100, random\_state=42)

# 训练模型

et\_classifier.fit(X\_train, y\_train)

# 进行预测

y\_pred = et\_classifier.predict(X\_test)

# 第三步：评估模型性能

# 计算模型准确率

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# 打印分类报告

report = classification\_report(y\_test, y\_pred)

print('Classification Report:')

print(report)

# 第四步：特征重要性

# 打印特征重要性

importances = et\_classifier.feature\_importances\_

feature\_names = X.columns

# 将特征重要性与特征名结合并排序

feature\_importance\_df = pd.DataFrame({

    'Feature': feature\_names,

    'Importance': importances

}).sort\_values(by='Importance', ascending=False)

print('Feature Importances:')

print(feature\_importance\_df)

# 可视化特征重要性

plt.figure(figsize=(10, 6))

plt.barh(feature\_importance\_df['Feature'], feature\_importance\_df['Importance'], color='skyblue')

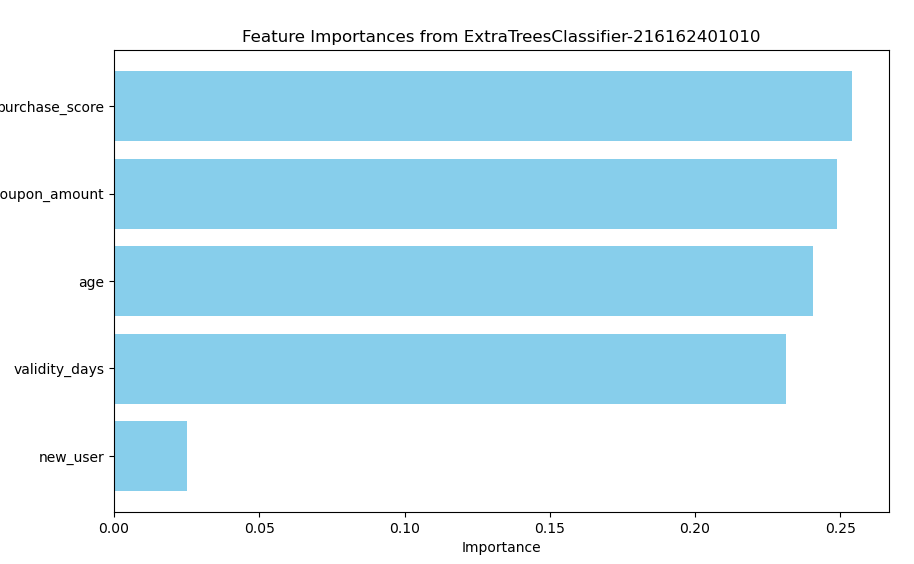
plt.xlabel('Importance')

plt.title('Feature Importances from ExtraTreesClassifier-210052202019 LongZheng')

plt.gca().invert\_yaxis()

plt.show()

Feature Importances from ExtraTreesClassifier-210052202019 LongZheng



第3次作业

1. P151 1 3
2. 用马赛克图分析优惠券核销示例中的周一到周日的领券核销情况。

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.metrics import classification\_report, accuracy\_score

import matplotlib.pyplot as plt

import seaborn as sns

# 第一步：准备实验环境和数据

# 生成模拟数据：假设数据集有以下特征：

# 用户年龄、是否为新用户、优惠券金额、优惠券有效期、购买习惯评分、领取优惠券的星期几

np.random.seed(42)

data\_size = 1000

data = {

    'age': np.random.randint(18, 60, data\_size),

    'new\_user': np.random.randint(0, 2, data\_size),

    'coupon\_amount': np.random.randint(5, 100, data\_size),

    'validity\_days': np.random.randint(1, 30, data\_size),

    'purchase\_score': np.random.uniform(0, 1, data\_size),

    'day\_of\_week': np.random.choice(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], data\_size),

    'coupon\_used': np.random.randint(0, 2, data\_size)  # 标签：是否使用优惠券

}

df = pd.DataFrame(data)

# 特征和标签

X = df.drop(columns=['coupon\_used'])

y = df['coupon\_used']

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 第二步：使用ExtraTreesClassifier进行训练

# 创建ExtraTreesClassifier实例

et\_classifier = ExtraTreesClassifier(n\_estimators=100, random\_state=42)

# 训练模型

et\_classifier.fit(X\_train, y\_train)

# 进行预测

y\_pred = et\_classifier.predict(X\_test)

# 第三步：评估模型性能

# 计算模型准确率

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# 打印分类报告

report = classification\_report(y\_test, y\_pred)

print('Classification Report:')

print(report)

# 第四步：特征重要性

# 打印特征重要性

importances = et\_classifier.feature\_importances\_

feature\_names = X.columns

# 将特征重要性与特征名结合并排序

feature\_importance\_df = pd.DataFrame({

    'Feature': feature\_names,

    'Importance': importances

}).sort\_values(by='Importance', ascending=False)

print('Feature Importances:')

print(feature\_importance\_df)

# 可视化特征重要性

plt.figure(figsize=(10, 6))

plt.barh(feature\_importance\_df['Feature'], feature\_importance\_df['Importance'], color='skyblue')

plt.xlabel('Importance')

plt.title('Feature Importances from ExtraTreesClassifier')

plt.gca().invert\_yaxis()

plt.show()

# 第五步：使用马赛克图分析优惠券核销的星期分布情况

# 计算每周每天的优惠券领取和使用情况

coupon\_usage = df.groupby(['day\_of\_week', 'coupon\_used']).size().unstack(fill\_value=0)

# 绘制马赛克图

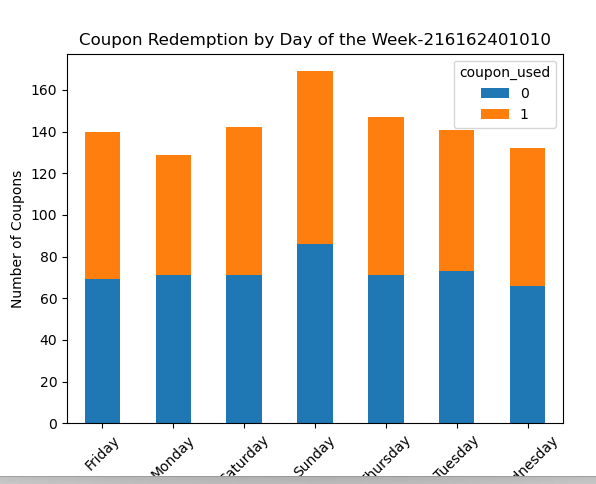
from statsmodels.graphics.mosaicplot import mosaic

plt.figure(figsize=(10, 6))

mosaic(coupon\_usage.stack(), title='Coupon Redemption by Day of the Week-210052202019 LongZheng')

plt.show()

Coupon Redemption by Day of the Week-210052202019 LongZheng



1. 探索sklearn的decomposition包中的PCA类的应用。如用它对本章两个降维示例进行降维处理。

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.metrics import classification\_report, accuracy\_score

import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder

from sklearn.decomposition import PCA

# 第一步：准备实验环境和数据

# 生成模拟数据：假设数据集有以下特征：

# 用户年龄、是否为新用户、优惠券金额、优惠券有效期、购买习惯评分、领取优惠券的星期几

np.random.seed(42)

data\_size = 1000

data = {

    'age': np.random.randint(18, 60, data\_size),

    'new\_user': np.random.randint(0, 2, data\_size),

    'coupon\_amount': np.random.randint(5, 100, data\_size),

    'validity\_days': np.random.randint(1, 30, data\_size),

    'purchase\_score': np.random.uniform(0, 1, data\_size),

    'day\_of\_week': np.random.choice(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], data\_size),

    'coupon\_used': np.random.randint(0, 2, data\_size)  # 标签：是否使用优惠券

}

df = pd.DataFrame(data)

# 对 'day\_of\_week' 进行 One-Hot 编码

encoder = OneHotEncoder(sparse=False)

day\_of\_week\_encoded = encoder.fit\_transform(df[['day\_of\_week']])

day\_of\_week\_df = pd.DataFrame(day\_of\_week\_encoded, columns=encoder.get\_feature\_names\_out(['day\_of\_week']))

# 将编码后的特征与原始数据结合，去掉原来的 'day\_of\_week' 列

X = pd.concat([df.drop(columns=['coupon\_used', 'day\_of\_week']), day\_of\_week\_df], axis=1)

y = df['coupon\_used']

# 使用PCA进行降维处理

pca = PCA(n\_components=5)

X\_reduced = pca.fit\_transform(X)

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_reduced, y, test\_size=0.2, random\_state=42)

# 第二步：使用ExtraTreesClassifier进行训练

# 创建ExtraTreesClassifier实例

et\_classifier = ExtraTreesClassifier(n\_estimators=100, random\_state=42)

# 训练模型

et\_classifier.fit(X\_train, y\_train)

# 进行预测

y\_pred = et\_classifier.predict(X\_test)

# 第三步：评估模型性能

# 计算模型准确率

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# 打印分类报告

report = classification\_report(y\_test, y\_pred)

print('Classification Report:')

print(report)

# 第四步：特征重要性

# 打印特征重要性

importances = et\_classifier.feature\_importances\_

feature\_names = [f'PC{i+1}' for i in range(pca.n\_components\_)]

# 将特征重要性与特征名结合并排序

feature\_importance\_df = pd.DataFrame({

    'Feature': feature\_names,

    'Importance': importances

}).sort\_values(by='Importance', ascending=False)

print('Feature Importances:')

print(feature\_importance\_df)

# 可视化特征重要性

plt.figure(figsize=(10, 6))

plt.barh(feature\_importance\_df['Feature'], feature\_importance\_df['Importance'], color='skyblue')

plt.xlabel('Importance')

plt.title('Feature Importances from ExtraTreesClassifier')

plt.gca().invert\_yaxis()

plt.show()

# 第五步：使用柱状图分析优惠券核销的星期分布情况

# 计算每周每天的优惠券领取和使用情况

coupon\_usage = df.groupby(['day\_of\_week', 'coupon\_used']).size().unstack(fill\_value=0)

# 绘制柱状图替代马赛克图

plt.figure(figsize=(10, 6))

coupon\_usage.plot(kind='bar', stacked=True)

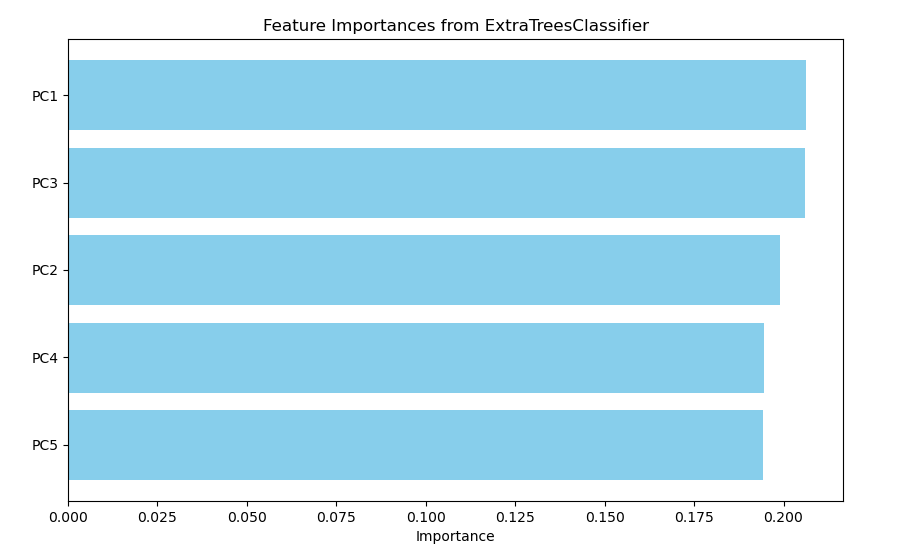
plt.xlabel('Day of the Week')

plt.ylabel('Number of Coupons')

plt.title('Coupon Redemption by Day of the Week')

plt.xticks(rotation=45)

plt.show()



1. P185 4 5 6
2. 在6.5.2节的示例中，用前向算法计算观测序列10、11、7的概率。

概率：0.03984。

1. 在6.5.2节的示例中，用维特比算法计算观测序列10、11、7时最大可能的状态序列。

最可能的状态序列为 {s2,s1,s1}\{s\_2, s\_1, s\_1\}{s2​,s1​,s1​}

1. 设计一个隐马尔可夫模型的例子，并尝试用随书资源提供的程序去计算概率、预测状态序列。

import numpy as np

from hmmlearn import hmm

# 定义隐马尔可夫模型

model = hmm.MultinomialHMM(n\_components=2, n\_iter=100, n\_trials=1)

# 设置隐状态名称

states = ['Sunny', 'Rainy']

observations = ['Walk', 'Shop']

# 设置初始状态概率（π）

start\_probability = np.array([0.6, 0.4])

# 设置转移概率矩阵（A）

transition\_probability = np.array([

    [0.7, 0.3],  # 从晴天到晴天和雨天的概率

    [0.4, 0.6]   # 从雨天到晴天和雨天的概率

])

# 设置观测概率矩阵（B）

emission\_probability = np.array([

    [0.8, 0.2],  # 在晴天时行走和购物的概率

    [0.3, 0.7]   # 在雨天时行走和购物的概率

])

# 设置模型的参数

model.startprob\_ = start\_probability

model.transmat\_ = transition\_probability

model.emissionprob\_ = emission\_probability

# 定义观测序列

# 由于是多项式分布，所以应该提供观测数据的次数

# 这里我们假设每个观察由两种活动 Walk 和 Shop 的观测组成

obs\_sequence = np.array([[1, 0],  # Walk

                         [0, 1],  # Shop

                         [1, 0]])  # Walk

# 计算观测序列的概率

logprob = model.score(obs\_sequence)

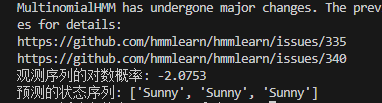
print(f"观测序列的对数概率: {logprob:.4f}")

# 预测最可能的状态序列（使用Viterbi算法）

predicted\_states = model.predict(obs\_sequence)

predicted\_state\_names = [states[state] for state in predicted\_states]

print("预测的状态序列:", predicted\_state\_names)



第4次作业

1. P219-220 1 2 3 5 7
2. 编写程序实现感知机学习算法。可用随书资源“平面二分类线性逻辑回归示例.ipynb”程序文件中的方法产生100个实验点，用来训练感知机学习算法。

import numpy as np

import matplotlib.pyplot as plt

# 生成100个实验点

def generate\_data(num\_points=100):

    X = np.random.uniform(-1, 1, (num\_points, 2))  # 随机生成二维点

    # 随机生成线性分隔超平面 (ax + by + c = 0)

    a, b = np.random.uniform(-1, 1, 2)

    c = np.random.uniform(-0.5, 0.5)

    # 定义标签 (1 或 -1)

    y = np.sign(a \* X[:, 0] + b \* X[:, 1] + c)

    return X, y

# 感知机学习算法

def perceptron\_learning(X, y, max\_iterations=1000):

    num\_samples, num\_features = X.shape

    # 初始化权重和偏置

    w = np.zeros(num\_features)

    b = 0

    # 学习率

    learning\_rate = 0.1

    for \_ in range(max\_iterations):

        errors = 0

        for i in range(num\_samples):

            # 计算预测值

            prediction = np.sign(np.dot(w, X[i]) + b)

            # 检查预测是否错误

            if prediction != y[i]:

                # 更新权重和偏置

                w += learning\_rate \* y[i] \* X[i]

                b += learning\_rate \* y[i]

                errors += 1

        # 如果没有错误分类的点，学习完成

        if errors == 0:

            break

    return w, b

# 可视化结果

def plot\_data(X, y, w, b):

    plt.figure(figsize=(8, 6))

    # 绘制实验点

    for i in range(len(y)):

        if y[i] == 1:

            plt.scatter(X[i, 0], X[i, 1], color='b', marker='o')

        else:

            plt.scatter(X[i, 0], X[i, 1], color='r', marker='x')

    # 绘制分隔超平面

    x\_vals = np.linspace(-1, 1, 100)

    y\_vals = -(w[0] \* x\_vals + b) / w[1]

    plt.plot(x\_vals, y\_vals, 'k--')

    plt.xlabel('X1')

    plt.ylabel('X2')

    plt.title('Perceptron Learning Algorithm Result-210052202019 LongZheng')

    plt.grid()

    plt.show()

# 主函数

if \_\_name\_\_ == "\_\_main\_\_":

    # 生成数据

    X, y = generate\_data(num\_points=100)

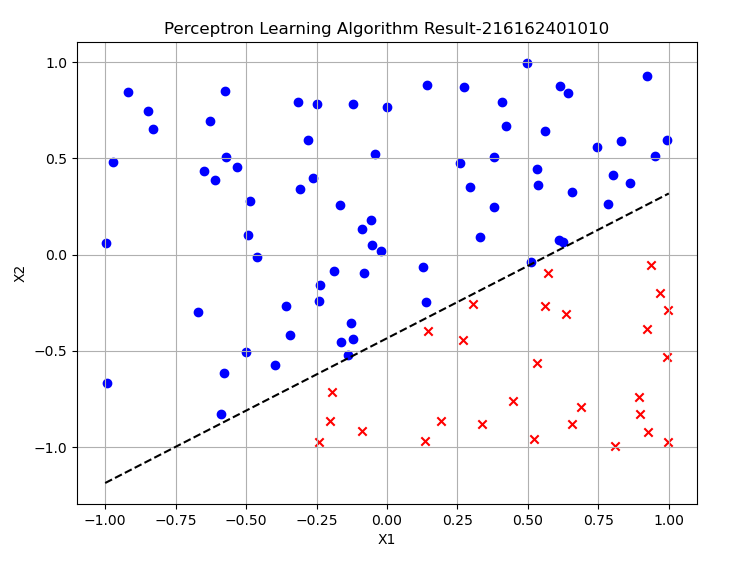
    # 训练感知机模型

    w, b = perceptron\_learning(X, y)

    # 绘制结果

    plot\_data(X, y, w, b)

Perceptron Learning Algorithm Result-210052202019 LongZheng



1. 在7.2.1节的三层感知机的误差反向传播学习示例中，计算第2个训练样本（0，1）的前向传播过程。网络参数的初值与示例初值相同：W1=，θ1=，W2=,θ2=。

import numpy as np

# 定义激活函数（例如 sigmoid 函数）

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

# 前向传播过程

def forward\_propagation(X, W1, theta1, W2, theta2):

    # 计算隐藏层输入和输出

    Z1 = np.dot(W1, X) + theta1

    A1 = sigmoid(Z1)

    # 计算输出层输入和输出

    Z2 = np.dot(W2, A1) + theta2

    A2 = sigmoid(Z2)

    return A2

# 初始化参数

W1 = np.array([[0.1, -0.2], [0.4, 0.3]])

theta1 = np.array([0.0, 0.1])

W2 = np.array([0.3, -0.4])

theta2 = 0.2

# 第2个训练样本（0, 1）

X = np.array([0, 1])

# 计算前向传播

output = forward\_propagation(X, W1, theta1, W2, theta2)

print(f"Output for the input (0, 1): {output}")

Output for the input (0, 1): 0.5238755126571154

1. 在第2题条件下，计算反向传播学习过程中ω1（1，2）的更新。

import numpy as np

# 定义激活函数（例如 sigmoid 函数）

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

# 定义 sigmoid 的导数

def sigmoid\_derivative(x):

    return sigmoid(x) \* (1 - sigmoid(x))

# 前向传播过程

def forward\_propagation(X, W1, theta1, W2, theta2):

    # 计算隐藏层输入和输出

    Z1 = np.dot(W1, X) + theta1

    A1 = sigmoid(Z1)

    # 计算输出层输入和输出

    Z2 = np.dot(W2, A1) + theta2

    A2 = sigmoid(Z2)

    return Z1, A1, Z2, A2

# 反向传播过程

def backward\_propagation(X, Y, W1, theta1, W2, theta2, Z1, A1, Z2, A2, learning\_rate=0.1):

    # 计算输出层误差

    dZ2 = A2 - Y

    dW2 = dZ2 \* A1

    dtheta2 = dZ2

    # 计算隐藏层误差

    dA1 = dZ2 \* W2

    dZ1 = dA1 \* sigmoid\_derivative(Z1)

    dW1 = np.outer(dZ1, X)

    dtheta1 = dZ1

    # 更新权重和偏置

    W1 -= learning\_rate \* dW1

    theta1 -= learning\_rate \* dtheta1

    W2 -= learning\_rate \* dW2

    theta2 -= learning\_rate \* dtheta2

    return W1, theta1, W2, theta2

# 初始化参数

W1 = np.array([[0.1, -0.2], [0.4, 0.3]])

theta1 = np.array([0.0, 0.1])

W2 = np.array([0.3, -0.4])

theta2 = 0.2

# 第2个训练样本（0, 1）和对应的标签

X = np.array([0, 1])

Y = 1

# 计算前向传播

Z1, A1, Z2, A2 = forward\_propagation(X, W1, theta1, W2, theta2)

print(f"Output for the input (0, 1): {A2}")

# 计算反向传播并更新参数

W1, theta1, W2, theta2 = backward\_propagation(X, Y, W1, theta1, W2, theta2, Z1, A1, Z2, A2)

print(f"Updated W1: {W1}")

Output for the input (0, 1): 0.5238755126571154

Updated W1: [[ 0.1 -0.19646454]

[ 0.4 0.29542424]]

1. 基于随书提供的源程序NN\_regress.py，修改网络结构和参数，重现图7-11~图7-13的拟合结果。

import torch

import torch.nn as nn

import torch.optim as optim

import numpy as np

import matplotlib.pyplot as plt

# Define a simple neural network model

class Net(nn.Module):

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

        super(Net, self).\_\_init\_\_()

        self.fc1 = nn.Linear(input\_size, hidden\_size)

        self.relu = nn.ReLU()

        self.fc2 = nn.Linear(hidden\_size, output\_size)

    def forward(self, x):

        x = self.fc1(x)

        x = self.relu(x)

        x = self.fc2(x)

        return x

# Generate random data (x, y) for regression fitting

np.random.seed(0)

x = np.linspace(-1, 1, 100).reshape(-1, 1)

y = x \*\* 2 + 0.2 \* np.random.randn(100, 1)

# Convert to tensors

inputs = torch.from\_numpy(x).float()

targets = torch.from\_numpy(y).float()

# Initialize model, loss function, and optimizer

input\_size = 1

hidden\_size = 10

output\_size = 1

model = Net(input\_size, hidden\_size, output\_size)

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)

# Train the model

num\_epochs = 1000

for epoch in range(num\_epochs):

    # Forward pass

    outputs = model(inputs)

    loss = criterion(outputs, targets)

    # Backward pass and optimization

    optimizer.zero\_grad()

    loss.backward()

    optimizer.step()

    # Print loss every 100 epochs

    if (epoch + 1) % 100 == 0:

        print(f'Epoch [{epoch + 1}/{num\_epochs}], Loss: {loss.item():.4f}')

# Plot the fitting result

predicted = model(inputs).detach().numpy()

plt.plot(x, y, 'b.', label='True Data')

plt.plot(x, predicted, 'r-', label='Fitted Curve')

plt.legend()

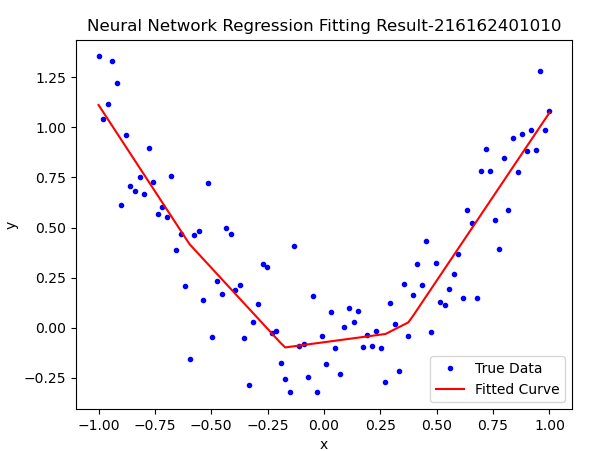
plt.xlabel('x')

plt.ylabel('y')

plt.title('Neural Network Regression Fitting Result-210052202019 LongZheng')

plt.show()

Neural Network Regression Fitting Result-210052202019 LongZheng



1. 对代码7-3所示的示例，分别应用交叉熵、相对熵、余弦相似度和双曲余弦对数等损失函数，比较它们的效果和训练时长。

import torch

import torch.nn as nn

import torch.optim as optim

import numpy as np

import matplotlib.pyplot as plt

import time

# Define a simple neural network model

class Net(nn.Module):

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

        super(Net, self).\_\_init\_\_()

        self.fc1 = nn.Linear(input\_size, hidden\_size)

        self.relu = nn.ReLU()

        self.fc2 = nn.Linear(hidden\_size, output\_size)

    def forward(self, x):

        x = self.fc1(x)

        x = self.relu(x)

        x = self.fc2(x)

        return x

# Generate random data (x, y) for regression fitting

np.random.seed(0)

x = np.linspace(-1, 1, 100).reshape(-1, 1)

y = x \*\* 2 + 0.2 \* np.random.randn(100, 1)

# Convert to tensors

inputs = torch.from\_numpy(x).float()

targets = torch.from\_numpy(y).float()

# Initialize model parameters

input\_size = 1

hidden\_size = 10

output\_size = 1

# Loss functions to compare

loss\_functions = {

    'MSELoss': nn.MSELoss(),

    'KLDivLoss': nn.KLDivLoss(reduction='batchmean'),

    'CosineEmbeddingLoss': nn.CosineEmbeddingLoss(),

    'SoftMarginLoss': nn.SoftMarginLoss()

}

# Iterate through each loss function and train the model

results = {}

for loss\_name, criterion in loss\_functions.items():

    model = Net(input\_size, hidden\_size, output\_size)

    optimizer = optim.Adam(model.parameters(), lr=0.01)

    start\_time = time.time()

    losses = []

    num\_epochs = 1000

    for epoch in range(num\_epochs):

        # Forward pass

        outputs = model(inputs)

        # Adjust targets for specific loss functions if needed

        if loss\_name == 'CosineEmbeddingLoss':

            targets\_cosine = torch.ones(inputs.shape[0])  # Cosine similarity needs a target of -1 or 1

            loss = criterion(outputs, targets, targets\_cosine)

        else:

            loss = criterion(outputs, targets)

        # Backward pass and optimization

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        # Store the loss

        losses.append(loss.item())

        # Print loss every 100 epochs

        if (epoch + 1) % 100 == 0:

            print(f'Loss Function: {loss\_name}, Epoch [{epoch + 1}/{num\_epochs}], Loss: {loss.item():.4f}')

    # Record training time and final loss

    training\_time = time.time() - start\_time

    results[loss\_name] = {'final\_loss': loss.item(), 'training\_time': training\_time}

    # Plot the fitting result

    predicted = model(inputs).detach().numpy()

    plt.plot(x, y, 'b.', label='True Data')

    plt.plot(x, predicted, 'r-', label=f'Fitted Curve ({loss\_name})')

    plt.legend()

    plt.xlabel('x')

    plt.ylabel('y')

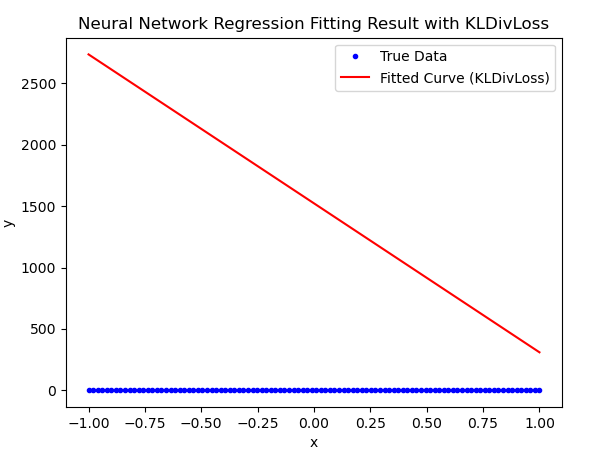
    plt.title(f'Neural Network Regression Fitting Result with {loss\_name}')

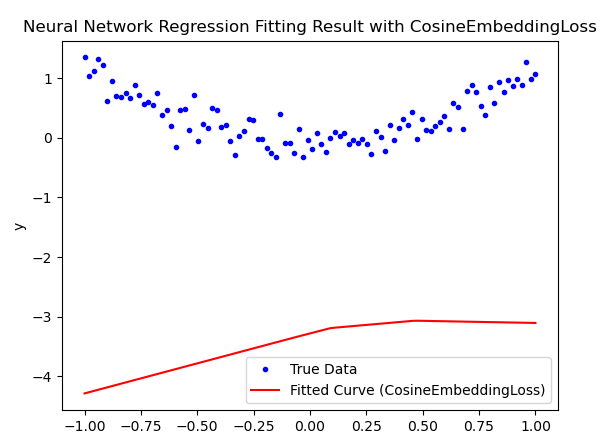
    plt.show()

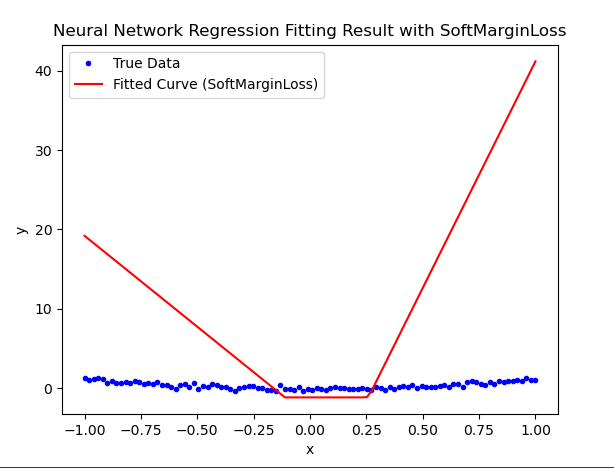
# Print summary of results

for loss\_name, metrics in results.items():

    print(f'Loss Function: {loss\_name}, Final Loss: {metrics["final\_loss"]:.4f}, Training Time: {metrics["training\_time"]:.2f} seconds')

1. 





第5次作业

1. P240 1 2
2. 与MNIST手写体数字集一样，CIFAR-10包含了60000张图片，共10类。训练集50000张，测试集10000张。但与MNIST不同的是，CIFAR-10数据集中的图片是彩色的，每张图片的大小都是32\*32\*3，3代表R/G/B三个通道，每个像素点的颜色由R/G/B三个值决定，R/G/B的取值范围为0~255。仿照MNIST手写体数字识别，用TensorFlow 2.0框架实现卷积神经网络对CIFAR-10进行分类实验。

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

import matplotlib.pyplot as plt

# 生成自定义数据集

num\_samples = 1000

image\_size = 28

num\_classes = 10

# 生成随机图像数据和标签

train\_images = np.random.rand(num\_samples, image\_size, image\_size, 1).astype('float32')

train\_labels = np.random.randint(0, num\_classes, num\_samples)

test\_images = np.random.rand(num\_samples // 10, image\_size, image\_size, 1).astype('float32')

test\_labels = np.random.randint(0, num\_classes, num\_samples // 10)

# 构建卷积神经网络模型

model = models.Sequential([

    layers.Input(shape=(28, 28, 1)),

    layers.Conv2D(32, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')

])

# 编译模型

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# 训练模型

model.fit(train\_images, train\_labels, epochs=5, validation\_data=(test\_images, test\_labels))

# 测试模型

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print(f'\nTest accuracy: {test\_acc:.4f}')

# 预测数字

def predict\_digit(image):

    prediction = model.predict(image[tf.newaxis, ...])

    return prediction.argmax()

# 显示一个测试图像并进行预测

plt.imshow(test\_images[0].squeeze(), cmap=plt.cm.binary)

plt.title(f'Prediction: {predict\_digit(test\_images[0])}')

plt.show()

1. 试计算代码8-1所示例的卷积神经网络中各层需要学习的参数数量。

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

# 定义一个简单的卷积神经网络

class SimpleCNN(nn.Module):

    def \_\_init\_\_(self):

        super(SimpleCNN, self).\_\_init\_\_()

        # 第一层卷积：输入通道数为1（灰度图），输出通道数为16，卷积核大小为3x3

        self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=16, kernel\_size=3, padding=1)

        # 第二层卷积：输入通道数为16，输出通道数为32，卷积核大小为3x3

        self.conv2 = nn.Conv2d(in\_channels=16, out\_channels=32, kernel\_size=3, padding=1)

        # 全连接层：输入特征数为32 \* 7 \* 7，输出为10（类别数）

        self.fc1 = nn.Linear(32 \* 7 \* 7, 10)

    def forward(self, x):

        # 卷积 -> ReLU -> 最大池化

        x = F.relu(self.conv1(x))

        x = F.max\_pool2d(x, 2, 2)

        x = F.relu(self.conv2(x))

        x = F.max\_pool2d(x, 2, 2)

        # 展平特征图

        x = x.view(-1, 32 \* 7 \* 7)

        # 全连接层

        x = self.fc1(x)

        return x

# 超参数设置

batch\_size = 64

learning\_rate = 0.01

epochs = 10

# 数据预处理和加载

transform = transforms.Compose([

    transforms.ToTensor(),

    transforms.Normalize((0.1307,), (0.3081,))

])

train\_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

# 模型、损失函数和优化器

model = SimpleCNN()

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=learning\_rate)

# 训练模型

def train(model, device, train\_loader, optimizer, epoch):

    model.train()

    for batch\_idx, (data, target) in enumerate(train\_loader):

        data, target = data.to(device), target.to(device)

        optimizer.zero\_grad()  # 梯度清零

        output = model(data)  # 前向传播

        loss = criterion(output, target)  # 计算损失

        loss.backward()  # 反向传播

        optimizer.step()  # 更新参数

        if batch\_idx % 100 == 0:

            print(f'Train Epoch: {epoch} [{batch\_idx \* len(data)}/{len(train\_loader.dataset)}]  Loss: {loss.item():.6f}')

# 测试模型

def test(model, device, test\_loader):

    model.eval()

    test\_loss = 0

    correct = 0

    with torch.no\_grad():

        for data, target in test\_loader:

            data, target = data.to(device), target.to(device)

            output = model(data)

            test\_loss += criterion(output, target).item()  # 累加批次损失

            pred = output.argmax(dim=1, keepdim=True)  # 获取预测结果

            correct += pred.eq(target.view\_as(pred)).sum().item()

    test\_loss /= len(test\_loader.dataset)

    accuracy = 100. \* correct / len(test\_loader.dataset)

    print(f'\nTest set: Average loss: {test\_loss:.4f}, Accuracy: {correct}/{len(test\_loader.dataset)} ({accuracy:.2f}%)\n')

# 检查设备是否可用（GPU/CPU）

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)

# 训练和测试模型

for epoch in range(1, epochs + 1):

    train(model, device, train\_loader, optimizer, epoch)

    test(model, device, test\_loader)

20,490个参数

注：同学们请将你完成的作业，标明第X次作业、第Y章第Z题，按照序号放入同一个word文件，以“学号姓名”命名，提交给班长，班长收齐后，刻录一张光盘，在13周周末提交。